METHOD AND COMPUTER SYSTEM FOR CUSTOMIZING COMPUTER APPLICATIONS BY STORING THE CUSTOMIZATION SPECIFICATIONS AS DATA IN A DATABASE

The present application claims priority from a provisional application entitled "Method and Computer System for Customizing Computer Applications by Storing the Customization Specifications as Data in a Database," Serial No. 60/225,075, filed on August 14, 2000 and now pending.

FIELD OF THE INVENTION

The present invention relates to a method and computer system to customize computer software, including multilingual labels, error messages, context-sensitive help, security, and client preferences, such as display preferences, by storing the customization specifications as data in a database.

BACKGROUND OF THE INVENTION

Typically, when users want a software application customized to meet their needs, it is necessary for application developers to use a programming language or software development environment to modify existing code. However, it is expensive in terms of both time and money for application developers to customize software. Consequently, it is advantageous if users could customize the applications themselves.

In particular, if an application is needed to be deployed in two or more languages, traditional software developers would write multiple versions of the same screen, causing the number of screens in the system to be increase by a factor of two or more. In a

traditional system, a different set of screens would be required to support each additional language. It is expensive to support multilingual requirements in this manner.

Error messages are another area that users would like to customize. Frequently, error messages are written in a technical way that an applications developer can understand, but not necessarily a user. There is a strong need in the industry to be able to provide users a tool that they themselves (or documentation writers) can use to write meaningful, user-friendly error messages.

Context-sensitive help is becoming a standard in the computer industry. End user applications are becoming increasingly documented with help screens. However, the effectiveness of help still could be greatly improved by giving users or documentation writers the power to write their own error messages.

The ability to support a sophisticated access control and security mechanism is another important aspect that can be thought of as customizing software. Different users who have different roles will have varying security requirements.

Finally, unique and different client preferences are another aspect that contribute to increasingly complex systems administration. Frequently, in computer systems the preferences are supported by generating multiple versions of the applications, each of which needs to be compiled and deployed appropriately throughout the organization. This creates an overall computing environment that is very unstable and difficult to support. In addition, users may not be able to express their display preferences until the testing phase, or even later such as when the entire system is in production. Consequently, there is a need for users to be able to customize their own desktops, including setting their own display preferences.

Today's programming languages and system development environments are complex, and mastered only by programmers, not by end users. Providing an environment or system in which users can specify their own customizations greatly increases the control that users have over their system, and users can then maintain their own customization information.

Given the above problems, the present invention provides novel methods and computer systems which give a user the capability to flexibly alter the run-time version of a program they are operating, while avoiding the shortcomings and drawbacks of prior methodologies.

SUMMARY OF THE INVENTION

The present invention enables users to specify a value for an attribute for any object that can be modified at run-time. The approach of only employing run-time modifications allows for only one application to be built and it can be used to support any number of different sets of users.

The system according to the present invention allows users to store the structure of their programs. It provides a repository to store the structure of programs.

The invention, in a further aspect, allows users to customize their software application to multiple languages.

In yet another aspect, the present invention provides a method and system to allow a user to customize their error messages.

The present invention further provides such systems that allow users to customize their own help messages, including context-sensitive help messages.

The present invention provides methods and systems to allows users to specify their own security requirements.

In another important aspect, this invention provides the capability for users to set their own display preferences.

Because the present invention models both the data and the data structure, a wide variety of functions, companies, and industries (such as, e.g., a retailer or tax organization) can implement the same system.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the invention, the following Detailed Description should be read in conjunction with the accompanying Drawings, wherein:

FIGURE 1A is a schematic representation (logical, entity-relationship diagram) of a simple Transaction and Transaction Detail example;

FIGURES 1B-1C are physical schematic representations of various Entities, the various respective information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURES 1D-1E are examples of sample data in various Entities and Tables shown in **FIGURES 1B-1C**;

FIGURE 2 is an overview of an embodiment of the present invention;

FIGURE 3A is the overall logical, entity-relationship diagram (a schematic diagram) of the relational database design (the data model) used in the construction of an illustrative embodiment of the overall application of the present invention;

FIGURE 3B is the detailed logical, entity-relationship diagram (a schematic diagram) of the relational database design (the data model) used in the construction of an illustrative embodiment of the overall application of the present invention;

FIGURES 4-A to 4-K are physical schematic representations of various Entities, the various respective information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURES 5-A to 5-N are graphical display representations which are used by the present invention to enter tools, label sets, and systems;

FIGURE. 6 is an example of a computer application which is used as the basis for the sample data to populate the tables in **FIGURES 7-A to 7-K**; and

FIGURES 7-A to 7-K are examples of sample data in the various entities and tables.

DETAILED DESCRIPTION OF THE INVENTION

The present invention is described with reference to one specific type of graphical data modeling, entity-relationship (ER) diagrams. However, those of ordinary skill in that art will recognize that the scope of the present invention is not limited to this particular embodiment.

This section first briefly describes how to interpret the ER diagrams, and their associated tables, as they are used in various Figures throughout this document. Entity-Relationship Diagrams are well-known in the art and have a specific, precise meaning to database designers and developers, and consist of two major parts: entities and relationships.

FIGURE 1A is an example of a simplistic Entity-Relationship Diagram.

An entity is a thing of interest. The rectangular box represents an entity, and the text string in the first line of the box is the name of the entity. The text string in the second line of the box, having a pound sign ("#"), is the primary key. The primary key is the unique identifier of data in the table. In the case of the ER diagram of **FIGURE 1A** the name of the entity is "Transaction," and "Transaction_CD" is the primary key. The text strings in the following lines represent attributes. There are two types of attributes, namely "not null" attributes (i.e., mandatory attributes) which must have information entered when a user enters information, (indicated with an asterisk ("*")); and nullable attributes (i.e., optional attributes) which do not need to have information entered (indicated with the letter "o"). In "Transaction," the attribute "Name_TX" is a mandatory attribute, while the attribute "Descr_TX" is optional.

Symbol	**Meaning**
#	Primary Key
*	Not Null Field (i.e., mandatory)
o	Nullable Field (i.e., optional)

Relationships are represented in the Figures by lines between entities. There are a few types of relationships. The specific line in the example in **FIGURE 1A** represents a one-to-many relationship. In a one-to-many relationship, the primary key of Transaction (i.e., Transaction_*CD)* becomes the foreign key of the many, Transaction_Dtl. The name of the foreign key is denoted by the name of the primary key with an "FK" suffix added, i.e., Transaction_CD_FK. In this example, it is interpreted as Transactions have one or

more Transaction details associated with them. The Transaction *has* Transaction Details associated with it; and Transaction Details are *for* a specific Transaction.

Each entity can be supported by a table in the database. The tables corresponding to the Transaction entity and Transaction_Dtl entity are the Transaction table in **FIGURE 1B** and Transaction_Dtl table in **FIGURE 1C**. (**FIGURE 1C** is a physical schematic representation of the Entity entitled Transaction_Dtl, the various information fields thereof, the type of information contained therein, and the relationships contained therein.) The first field in the table is the attribute name. The second field is the sequence number for the attribute. The third field is a Boolean value indicating whether or not the attribute is optional. The fourth field is the physical format of the attribute; the fifth field is the length of the attribute; and the sixth field is the number of decimal places.

Some sample data for *Transaction* and for *Transaction_Detail* are shown in **FIGURES 1D** and **1E**. The *column headings* in **FIGURES 1D** and **1E** are the *attributes* in the Entity-Relationship Diagrams. In the Transaction table, there are two types of transactions; their names are "buy product transaction," and "sell product transaction." The *Transaction_Detail* table consists of three details. The first transaction detail is that a product was bought on November 1, 1999; the second transaction detail is that a product was bought on November 5, 1999; and the third transaction detail is that a product was sold on November 1, 1999.

A preferred embodiment of the invention is described here, with reference to **FIGURES 2** through **5**.

FIGURE 2 is an overview of an embodiment of the present invention. In general, the system of the present invention (hereinafter "the Customization System") may be realized in a variety of ways depending on the enabling technology available at the time of realization, and particular application requirements at hand. In the illustrative embodiment, the System is realized as a decentralized network of computers, but it could also be implemented in a centralized computing environment.

As shown in **FIGURE 2**, a computer system **100** includes a rule engine **102** which enforces the business rules stored in the data model **104**. The application of the rule engine **102** using the data model **104** to a user's input **106** produces a database **108**.

FIGURE 3A is a logical, entity-relationship diagram (a schematic diagram) of the relational database design used in the construction of an illustrative embodiment of the overall application of the present invention. **FIGURE 3A** shows the entities and the relationships between the entities, and the important characteristics and advantages of the design. **FIGURE 3B** describes, in detail, the attributes that are associated with each entity.

A *tool* allows a person to identify the software products or development environments that are to be supported, in the entity Tool (Tool). Each tool represents a different software development product used to build a program. The significance of this design is that it allows a person to maintain user preference information about a number of products.

A *defined object* allows a person to store information about a product element of any tool, that is, any software product or development environment, in the entity Defined Object (Def_Obj). A defined object is a type of element or component for a given tool. Each defined object represents a different component or element, for a given tool. The

defined objects can be associated together or structured in a hierarchical manner. Together, tool and defined object allow a person to describe the types of objects and their structure, and have that information stored in a database, so that the defined objects and their structure can then be manipulated according to the usual database operators.

A *system* allows a person to specify a logical collection of software applications, in the entity called System (Systm). A system consists of a set of programs potentially spanning multiple tools gathered together in a logical whole. Each system represents a different complete software application that is supported.

A *valid tool* allows a person to identify which tool has been or could be used to develop a particular system, in the entity Valid Tool (Valid_Tool). The significance of this design is that it allows a person to specify what are the valid tools associated with a particular system. Different systems can be developed using different tools.

An *object* allows a person to specify an actual component of an actual system listed in the system entity, in the entity Object (Obj). Similarly to defined objects, objects can be classified in a hierarchical structure. Each object allows a person to specify actual components of systems and how they are structured hierarchically.

An *abstract object* allows a person to specify a generic object, in the entity Abstract Object (Abstr_Obj). An abstract object can be specified when a person wants to group together objects which have similar characteristics to each other. However, those objects may not have the same name in the different contexts, and there may be no easy way to identify that these objects all represent the same logical structure. To achieve this, the user declares that all of the objects associated with an object are attached to the same abstract object. By grouping them together in abstract object, a person can declare the

value of an attribute for an abstract object and it will automatically be inherited by every one of its associated objects.

A *language* allows a person to specify the languages that are permissible or can be supported, in the entity Language (Lang). The *label set* allows a person to specify which languages are associated with a particular system, in the entity Label Set (Label_Set). The significance of this entity is that is allows the labels in the user interface to appear in multiple languages, but with a single set of screens. In traditional system design, a second set of screens would be required to support a second language.

A *domain* allows a person to define the set of values from which an attribute may be drawn, in the entity Domain (Domain). A domain is a complex data type, that is, it is more than a simple data type such as string or number.

A *defined object attribute* allows a person to specify which attributes are associated with which defined object or abstract object, and its associated domain, in the entity Defined Object Attribute (Def_Obj_Attr). The significance of this entity is that it allows a user to describe the attributes which determine how the defined objects or abstract objects can be manipulated.

Object value allows a user to specify what is the value of a particular attribute in a particular context, for a particular object, defined object, or abstract object, in the entity Object Value (Obj_Value). Object value supports a kind of object orientation so that a user can set the property globally by associating the value with the defined object. If there is a value for the same object at the abstract level, this value would override the defined level. The object level value will override both the abstract and defined levels.

This repository was initially designed to support multilingual labels. The label set entity stores which language is associated with a particular system. The values of the different labels are stored in object value. In other words, the different names of the labels in the different languages are stored in the object value entity and table.

Even though this repository was originally designed to support multilingual labels, it can also be used for other tasks as well, because in supporting multilingual labels, it also supports run-time modification of any attribute on any object in a program. One particularly useful application of this technology is in supporting help messages. Help messages are textual strings, which can be accessed while an application is running. These messages are context-sensitive, meaning that depending upon the object in the application, a different help message may be appropriate. In the context of the present invention, the help text can be viewed as simply another attribute, just like the label, or a particular object thus their repository is able to support storage of help message text without modification. Help text is stored in object value, with a defined object type of help text. This mechanism is particularly powerful since the help text is no longer embedded in application code. Therefore, help text data entry can be more safely assigned to non-technical users.

Error messages are similar to help messages, in that these messages are represented as another text field. With help messages, the help message was simply an additional attribute, which was then associated with existing system objects in the object table. Help was defined for specific things, such as the employee block, and not the block in general. With error messages, the process is somewhat different.

For error messages, it is necessary to create a new defined object, namely, that of an error event. There will be a separate error event for each unique error exception in the system. Error events may be defined system level, or, likely any other object, may be defined as residing recursively under any other object. The error text is stored in the object value table, just like all other values. Similarly to the help message procedure, non-technical users can maintain the actual text of error messages.

Not only can the architecture of the Customization System of the present invention support multilingual labels and unique textual messages, it can also support security. Frequently, different classes of users need to have different access to different applications, or different capabilities with respect to applications. There are four aspects to security that this system supports. First, there is application access, i.e. access to the programs themselves. Second, there is field-level security, which is the visibility or capability to edit specific fields in the application. Third, there is program unit control, which is the ability to execute any function in the system. Fourth, there is the ability to control the passing of parameters to the application which the program then uses to programmatically alter its behavior.

These four mechanisms support a rich security environment with all the necessary information stored in the repository. In its basic form, the implementation of application security is much the same as the other areas previously described, in that at run-time, users can change the properties of objects. For example, if users want to change a field's label, they simply change the label property of the field. If users want to make the field invisible or editable, they simply change the value of the invisible or updatable property. The security areas each have to be handled using similar, but slightly different,

mechanisms. To control access to the applications, users can disable the buttons that invoke those applications. Then, field-level security is as described above. Program unit access is handled by creating a new defined object called security event. For each place in the application where users want to check security, they can create another object, and enter a value of true or false in the label set. The method of implementation in some products is that developers write a single line of code called check security where they pass the name of the security object to the function, which automatically aborts the procedure resulting in an error message if security is inadequate. For number four, passing parameters is very simple. The security parameter is just another type of direct object, and the values are stored in the value table in the usual way.

Finally, the design and architecture of the Customization System of the present invention also support user preferences, including display preferences. Each field and label on the display can be customized, as each field can have its own unique individual preferences, that is, visual attributes, such as fonts, color, background, height of all buttons or other visual attributes.

Specification Of The Information Structures Comprising The Database Of The Present Invention

FIGURES 4-A to 4-K are physical schematic representations of various Entities, the various respective information fields thereof, the type of information contained therein, and the relationships contained therein.

FIGURE 4-A is a physical schematic representation of the Entity entitled Tool, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-B is a physical schematic representation of the Entity entitled Def_Obj, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-C is a physical schematic representation of the Entity entitled Systm, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-D is a physical schematic representation of the Entity entitled Valid_Tool, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-E is a physical schematic representation of the Entity entitled Obj, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-F is a physical schematic representation of the Entity entitled Abstr_Obj, the various information fields thereof, the type of information contained therein, and the relationships contained therein;

FIGURE 4-G is a physical schematic representation of the Entity entitled Lang, the various information fields thereof, the type of information contained therein, and the relationships contained therein; and

FIGURE 4-H is a physical schematic representation of the Entity entitled Label_Set, the various information fields thereof, the type of information contained therein, and the relationships contained therein; and

FIGURE 4-I is a physical schematic representation of the Entity entitled Domain, the various information fields thereof, the type of information contained therein, and the relationships contained therein.

FIGURE 4-J is a physical schematic representation of the Entity entitled Def_Obj_Attr, the various information fields thereof, the type of information contained therein, and the relationships contained therein.

FIGURE 4-K is a physical schematic representation of the Entity entitled Obj_Value, the various information fields thereof, the type of information contained therein, and the relationships contained therein.

With reference to **FIGURE 4A**, the user defines the tool by specifying the following attributes in a database table. First, each tool needs to have a unique identifier. This is done by specifying Tool_CD as the primary key. The system automatically generates the value for this field, and does so for all primary keys throughout the entire invention. Second, the user has the option of specifying the name of the rule in the attribute Name_TX. The user also has the option of specifying a description of the attribute in Descr_TX.

With reference to **FIGURE 4B**, the user identifies the defined objects by specifying the following attributes in a database table. First, each defined object needs to have a unique identifier. This is done by specifying Def_Obj_ID as the primary key. The user then has the option of specifying the name of the defined object in the attribute Name_TX. Finally, the user has the option of specifying a description of the defined object in the attribute Descr_TX.

With reference to **FIGURE 4C**, the user defines a system by specifying the following attributes in a database table. First, each system needs to have a unique identifier. This is done by specifying Systm_ID as the primary key. The user has the option of specifying a name of the system, in the attribute Name_TX. The user then has the option of specifying a description of the system, in the attribute Descr_TX. Next, the user can specify the password of the system, in Password_NR. Subsequently, the user can specify the schema of the system, in the attribute Schema_CD, and the version of the system, in Version_NR.

With reference to **FIGURE 4D**, the user specifies which tool is associated with which system by specifying the following attributes in a database table. First, each valid

tool needs to have a unique identifier. This is done by specifying Valid_Tool_ID as the primary key. The user specifies the tool, by using Tool_CD as a foreign key from Tool, resulting in Tool_CD_FK. The user than specifies the system, by using Systm_ID as a foreign key from Systm, resulting in Systm_ID_FK. The user has the option of specifying when the valid tool was created, in the attribute Creat_DT, and the person the valid tool was created by, in the attribute Creat_BY.

With reference to **FIGURE 4E**, the user specifies which defined object is used with which system, by specifying the following attributes in a database table. First, each object needs a unique identifier. This is done by specifying Obj_ID as the primary key. The user then has the option of specifying the name of the object, in Name_TX. The user then has the option of specifying a description of the object, in Descr_TX. The user then specifies the system, by using Systm_ID as a foreign key from Systm, resulting in Systm_ID_FK. Finally, the user specifies which defined object is relevant, by using Def_O b a r

With reference to **FIGURE 4F**, the user determines an abstract object, by specifying the following attributes in a database table. First, each abstract object needs to have a unique identifier. This is done by specifying Abstr_Obj_ID as the primary key. The user can then specify the name of the abstract object, in the attribute Name_TX. Next, the user has the option of specifying a description, in the attribute Descr_TX. Finally, the user specifies which system the abstract object is associated with, by using Systm_ID as a foreign key from Systm, resulting in Systm_ID_FK.

With reference to **FIGURE 4G**, the user identifies the language by specifying the following attributes in a database table. First, each language needs to have a unique

identifier. This is done by specifying Lang_CD as the primary key. The user then has the option of specifying the name of the language, in the attribute Name_TX. Finally, the user can specify a description of the language, in the attribute Descr_TX.

With reference to **FIGURE 4H**, the user specifies which language is used with which system, by specifying the following attributes in a database table. First, each label set needs to have a unique identifier. This is done by specifying Label_Set_CD as the primary key. The user then has the option of specifying the name of the label set, in Name_TX. The user then has the option of specifying a description of the label set, in Descr_TX. The user then specifies the system, by using Systm_ID as a foreign key from Systm, resulting in Systm_ID_FK. Finally, the user specifies which language is relevant, by using Lang_CD as a foreign key from Lang, resulting in Lang_CD_FK.

With reference to **FIGURE 4I**, the user determines the domain by specifying the following attributes in a database table. First, each domain needs to have a unique identifier. This is done by specifying Domain_CD as the primary key. The user then has the option of specifying the name of the domain, in the attribute Name_TX. The user can then specify a description of the domain, in the attribute Descr_TX. Next, the user has the option of entering what the data type of the domain is, in the attribute Data_Type_CD. Next, the user has the option of entering a default date, a default number, or a default text, in the attributes Deflt_DT, Deflt_NR, or Deflt_TX, respectively. The user can then enter the length of the domain, in the attribute Lngth_NR. Next, the user can specify the precision of the domain, in the attribute Prcsn_NR. The user then has the option of specifying a minimum value and a maximum value for the domain, in the attributes Min_NR and Max_NR, respectively. Next, the

user can specify the type of domain, in the attribute Z_Domain_CD. Finally, the user has the option to enter a valid rule, in the attribute Valid_Rule_TX.

With reference to **FIGURE 4J**, the user then identifies which defined objects are associated with which domains, by specifying the following attributes in a database table, Def_Obj_Attr. First, each defined object attribute needs to have a unique identifier. This is done by specifying the attribute Def_Obj_Attr_ID to be the primary key. The user can then specify the name of the defined object attribute, in the attribute Name_TX. Next, the user has the option of entering a description of the defined object attribute, in Descr_TX. The user can then enter an xxx in the attribute Abst_Obj_IDF_YN. Next, the user then specifies the defined object of interest, by using Def_Obj_ID as a foreign key from Def_Obj, resulting in Def_Obj_ID_FK. The user then specifies the domain of that defined object attribute, by using Domain_CD as a foreign key from Domain, resulting in Domain_CD_FK.

With reference to **FIGURE 4K**, the user then determines the value of the object, by specifying the following attributes in a database table, Obj_Val. First, each object value needs to have a unique identifier. This is done by specifying the attribute Obj_Val_ID to be the primary key. Next, the user has the option of specifying a numeric value, in the attribute Value_NR. The user then has the option of specifying a text value, in the attribute Value_TX. The user then determines which object the value is associated with, by specifying either the object, defined object, or the abstract object, by using Obj_ID, Def_Obj_ID, and Abstr_Obj_ID as foreign keys from Obj, Def_Obj_Attr, and Abstr_Obj, resulting in Obj_ID_FK, Def_Obj_ID_FK, and Abstr_Obj_ID_FK, respectively.

The significance of these eleven entities is that they are sufficient to capture and represent any data structure business rules, as well as other types of business rules, at a high level of abstraction, with a corresponding depth of flexibility and increase in design, development, and implementation efficiency. The repository, as specified in **FIGURES 4A – 4K**, store multilingual labels, error messages, context-sensitive help messages, security, and client preferences.

Graphical Display Of The Screens Supporting The Information Structures Comprising The Database Of The Present Invention

FIGURE 5-A to **5-N** show a graphical display by which users can enter information related to tools, labels sets, and systems. More specifically,

FIGURE 5-A is a graphical display to enter information into the entity and table called Tool;

FIGURE 5-B is a graphical display to view information of the entity and table called Defined Object;

FIGURE 5-C is a graphical display to enter information into the entities and tables called Defined Object and Defined Object Attribute;

FIGURE 5-D is a graphical display to enter information into the entity and table called Label Set;

FIGURE 5-E is a graphical display to enter information into the entities and tables called Object and Object Values;

FIGURE 5-F is a graphical display to enter information into the entities and tables called Defined Object and Object Values;

FIGURE 5-G is a graphical display to enter information into the entities and tables called Abstract Object and Object Values;

FIGURE 5-H is a graphical display to enter information into the entity and table called System;

FIGURE 5-I is a graphical display to enter language information into the entity and table called Label Set;

FIGURE 5-J is a graphical display to enter security information into the entities and tables called Label Set and Security Object Role;

FIGURE 5-K is a graphical display to enter information into the entity and table called Valid Tool;

FIGURE 5-L is a graphical display to enter information into the entities and tables called Abstract Object and Abstract Object Values;

FIGURE 5-M is a graphical display to view information of the entity called Object;

FIGURE 5-N is a graphical display to enter information into the entities and tables called Object and Object Values;

With reference to **FIGURE 5A**, the user can enter information that is stored in the entity, Tool. The user can enter a code in the field called Code, the name of the tool in the field called Tool, and a description in the field called Descr.

With reference to **FIGURE 5B**, the user can view the hierarchical information that is stored in the Defined Object entity, for a particular tool.

With reference to **FIGURE 5C**, the user can enter the name of the defined object, in the field called Name, that is associated with a particular tool. The hierarchical information is stored in the following manner: The Defined Object (Name) of Level N is the parent of all the Defined Object (Names) of Level N+1. In addition, the user can enter the attributes which describe a particular defined object, for a particular tool, in the bottom half of the screen.

With reference to **FIGURE 5D**, the user can enter information that is stored in the entity Label_Set. The user can enter the system ID in the field Systm_ID, a code in the field CD, the language in the field Lang, the name of the label set in the field Name, and a description of the label set in Descr.

With reference to **FIGURE 5E**, the user can enter information regarding a particular object, including the Tool_CD, ID, Def_Object, Name, and Descr. The user can also enter the values in the bottom half of the screen, in ID, Label Set / Language, Tool / Defined Object / Attribute, and Value.

With reference to **FIGURE 5F,** the user can display information regarding a particular defined object, including its hierarchical structure, and the attributes and values associated with each defined object, in the bottom half of the screen.

With reference to **FIGURE 5G,** the user can display information regarding a particular abstract object, and enter information in the attributes and values associated with each abstract object, in the bottom half of the screen.

With reference to **FIGURE 5H,** the user enters information regarding a particular system, in the entity System.

With reference to **FIGURE 5I,** the user enters information regarding the language label set, in the entity Label Set.

With reference to **FIGURE 5J,** the user enters information regarding the security label set, in the entities Label Set and Security Object Role.

With reference to **FIGURE 5K,** the user enters information regarding the valid tool, in the entity Valid Tool.

With reference to **FIGURE 5L,** the user enters information regarding the abstract object and the attributes and values associated with that abstract object, in the entities Abstract Object and Abstract Object Values.

With reference to **FIGURE 5M,** the user views information regarding the defined object and its hierarchical structure, in the entity Object.

With reference to **FIGURE 5N,** the user can enter information regarding the defined object and its hierarchical structure, as well as the values associated with it, in the entities Object and Object Values.

METHOD OF USING THE PRESENT INVENTION

The following example illustrates how this process and computer system can be used in practice, using sample data. **FIGURE 6** is an example of an application called Function Maintenance. **FIGURES 7-A** to **7-K** are examples of sample data in various of the entities and tables. More specifically,

FIG. **6** is the Function Maintenance software application which is used as the basis for the sample data which populates the tables in **FIGURES 7-A** to **7-K.**

FIG. **7A** is an example of sample data in the Entity and Table entitled Tool;

FIG. **7B** is an example of sample data in the Entity and Table entitled Def_Obj;

FIG. **7C** is an example of sample data in the Entity and Table entitled Systm;

FIG. **7D** is an example of sample data in the Entity and Table entitled Valid_Tool;

FIG. **7E** is an example of sample data in the Entity and Table entitled Obj;

FIG. **7F** is an example of sample data in the Entity and Table entitled Abstr_Obj;

FIG. **7G** is an example of sample data in the Entity and Table entitled Lang;

FIG. **7H** is an example of sample data in the Entity and Table entitled Label_Set;

FIG. **7I** is an example of sample data in the Entity and Table entitled Domain;

FIG. **7J** is an example of sample data in the Entity and Table entitled Def_Obj_Attr; and

FIG. **7K** is an example of sample data in the Entity and Table entitled Obj_Value.

With reference to **FIGURE 7A**, the user has entered software application development products, that is, three tools: Oracle Forms, Oracle Reports, and Microsoft SQL Server.

With reference to **FIGURE 7B**, the user has entered eleven defined objects, that is, eleven structural components of Oracle Forms, including forms, blocks, windows, canvases, tab pages, visual attributes, text items, display items, checkboxes, buttons, and radio groups. Contained within a form are blocks that include items. Forms also contain canvases and alerts. There are appropriate attributes associated with each of these types of objects. All of this information is stored in the first part of the repository, allowing it to work, not only for Oracle Forms, but also for any product used to generate forms. The only assumption is that the objects within the application are structured hierarchically.

With reference to **FIGURE 7C**, the user has entered three systems: the Payroll System, the system called ABC Corp., and the Marketing Information System.

With reference to **FIGURE 7D,** there are three valid tools: Forms for the Payroll System, Forms for ABC Corp. System, and Reports for ABC Corp. System.

With reference to **FIGURE 7E,** there are many objects. There are many names of each object, and each object is of a particular type, which is listed explicitly in the description field.

With reference to **FIGURE 7F**, there are three abstract objects.

With reference to **FIGURE 7G**, there are three languages, that is, three sets of multilingual labels that the system consists of: English, French, and German.

With reference to **FIGURE 7H**, the user has specified three languages for the payroll system, English, French and German.

With reference to **FIGURE 7I**, the user has specified two domains: money and small amount, which are of a numeric data type.